**Ai Report**

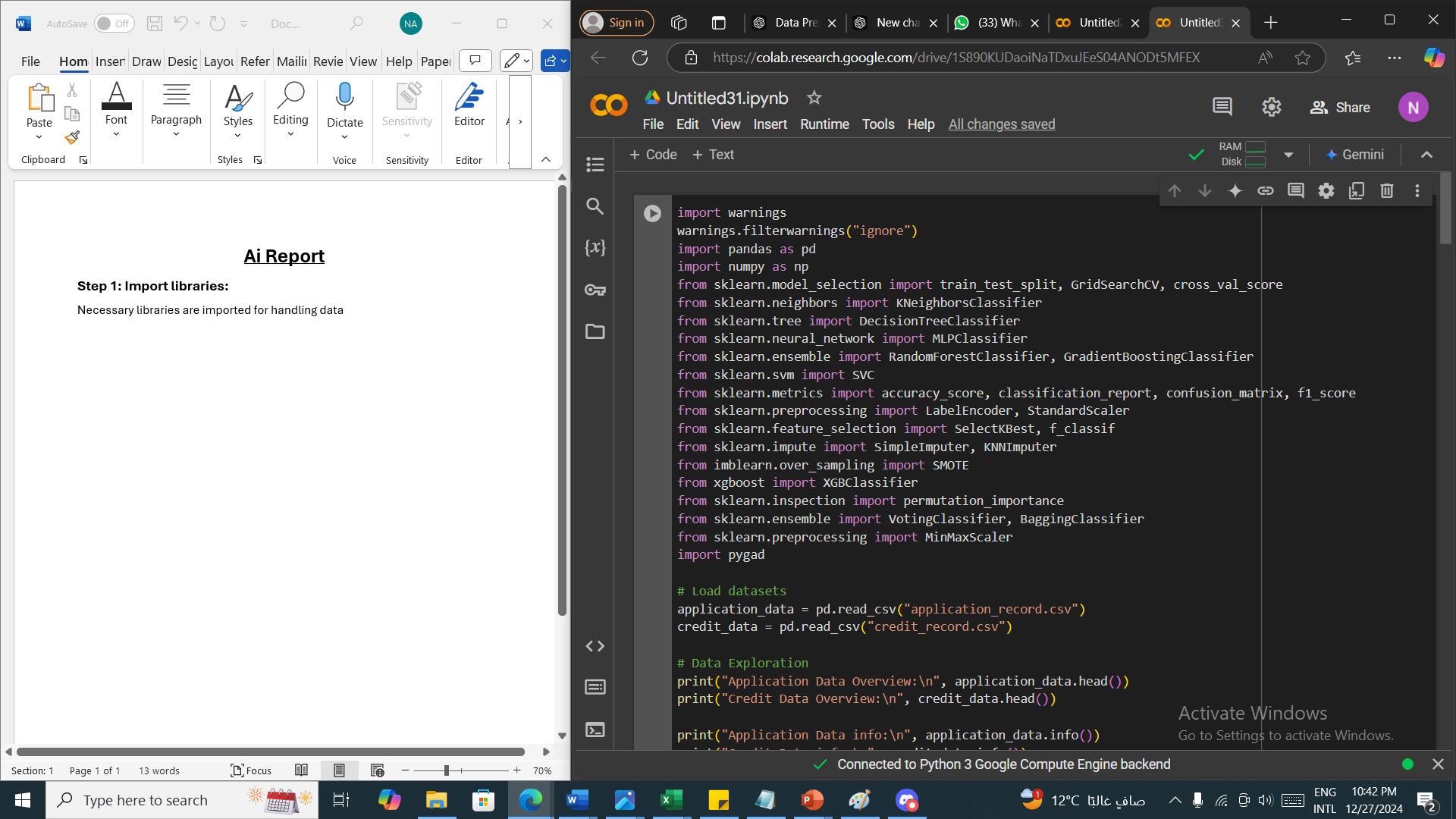
**Nourhan amr 2022/05080**

**Noor Hatem 2022/06987**

**Shaden essam 2022/08768**

**Abdullah mohamed 2022/07019**

**Step 1: Import libraries:**

Necessary libraries are imported for handling data

A screenshot of a computer

Description automatically generated**Step 2: Data Exploration**

* Load Datasets: The datasets application\_record.csv and credit\_record.csv are loaded into application\_data and credit\_data DataFrames using pandas.read\_csv().

A screenshot of a computer

Description automatically generated

* Data exploration:

1. .head() Shows the first 5 rows of each dataset for an initial look at the data.
2. .info() provides an overview of the dataset structure including column names , data types.
3. Isnull().sum() gets the sum for missing null values in each column

A screenshot of a computer

Description automatically generated**Step 3: Preprocessing**

* Handling Missing Data: identify and drop columns with missing values, The code finds the column with the highest percentage of missing values using **isnull().sum().idxmax().** This column is dropped because it likely has too much missing data to be useful. (OCCUPATION\_TYPE)

A screenshot of a computer screen

Description automatically generated

* A screenshot of a computer

  Description automatically generated**Removing Duplicate Rows**: duplicates() is used to remove rows that appear more than once. This ensures the data is clean and doesn't bias the model.
* **Imputing Missing Values for Numerical Columns:** Missing values in the specified numerical columns are filled with the median value (fillna(median\_value)). This is done because the median is more robust to outliers compared to the mean.

A screenshot of a computer screen

Description automatically generated

* **Outlier Detection and Clipping** **Using** **IQR Method**: The function wisker() calculates the lower (lw) and upper (uw) bounds using the Interquartile Range (IQR). Any values below lw or above uw are considered outliers. **Clipping**: Outliers are replaced by the nearest boundary (lw or uw) using np.where(). This prevents the model from being skewed by extreme values.

A screenshot of a computer

Description automatically generated

* **Mapping the STATUS Column**: The STATUS column in credit\_data contains categorical values, which we want to map to numerical values for better processing and model compatibility. We use a dictionary status\_mapping to map the values in the STATUS column to integers:

'C' and 'X' are mapped to 0, indicating a approved.'0', '1', '2', '3', '4', and '5' are mapped to 1, indicating not approved.

* **Aggregating the STATUS Values by ID**: Since Credit\_data has multiple rows per ID we need to aggregate the data by ID.The aggregation takes the **maximum** value of STATUS for each ID. After grouping, we reset the index to return a clean DataFrame.
* **Merging the Datasets**: After the aggregation, we merge application\_data and the modified credit\_data on the ID column using an **inner join** to ensure that only the matching rows are retained.

A screenshot of a computer

Description automatically generated

* **Label Encoding for Categorical Data:** Label Encoding: Categorical features are converted into numerical values using LabelEncoder. The fit\_transform() method assigns unique numeric values to each category.

A screenshot of a computer

Description automatically generated

* **Feature-Target Separation:** Features (X) and Target (y): The dataset is split into features (X) and the target variable (y). The target is STATUS, and features include all other columns except ID and STATUS.
* **Class Distribution:** The value\_counts() function is used to check the distribution of classes in the target variable. This helps identify if the classes are imbalanced.

A screenshot of a computer

Description automatically generated

* **Feature Scaling using Standardization**: Feature scaling is performed using StandardScaler, which normalizes the features by subtracting the mean and dividing by the standard deviation. This helps many models perform better (especially SVM, KNN, and neural networks).

**Step 3: Spliting Data into train, test and validation**

A screenshot of a computer

Description automatically generated

* **Train-Validation-Test Split:** First, the data is split into train (70%) and test (30%) sets, with stratify=y ensuring that both classes are represented proportionally.Then, the test set is further split into validation (15%) and test (15%) sets.

A screenshot of a computer

Description automatically generated**Step 4: Feature Selection (genetic algorithm)**

* **Fitness Function:** A genetic algorithm is used to select the best set of features by evaluating a fitness function. The solution is a binary array (chromosome) that indicates which features are selected.

A screenshot of a computer

Description automatically generatedThe fitness function computes the accuracy of a model (here, a Decision Tree) trained on the selected features using model.score() on the validation data.

* **pygad.GA()**:This initializes the **Genetic Algorithm (GA)** from the pygad library, which is used for feature selection in this case. Genetic algorithms are inspired by natural selection and work by evolving a population of candidate solutions over several generations.
* **Parameters**:num\_generations: The number of generations the algorithm will run (20 generations here). Each generation involves selecting parents, applying crossover, and mutation to produce the next generation.num\_parents\_mating: The number of parents that will be selected for mating in each generation (5 in this case).fitness\_func: The function that evaluates how good each solution is. This function is crucial for guiding the evolution towards better solutions. The fitness\_func used here is defined separately and is responsible for selecting the features based on accuracy.
* sol\_per\_pop: The number of solutions (or individuals) in each population (10 solutions in this case). Each solution represents a potential feature subset.
* num\_genes: The number of genes in a solution, which is equal to the number of features in X\_train (i.e., the number of features in your dataset).
* gene\_type: The data type of each gene (binary, i.e., int in this case).
* init\_range\_low and init\_range\_high: These set the range of possible values for each gene. Since it's a binary problem (feature selection: include or exclude), values are set between 0 and 2. The init\_range\_high is exclusive, so the possible values are 0 (exclude) and 1 (include).
* **Crossover and Mutation**:
* crossover\_type="single\_point": The method used to combine two parent solutions to produce a child solution. "Single-point" means a random crossover point is chosen, and the two parents exchange parts of their chromosomes at that point.
* crossover\_probability=0.9: The probability that crossover will happen (90% chance here).
* mutation\_type="random": Mutation randomly changes some of the genes (features) in a solution to explore new possibilities.
* mutation\_probability=0.1: The probability of mutation occurring (10% chance).
* mutation\_percent\_genes=10: Percentage of genes (features) that will be mutated.
* **Running the GA**:ga\_instance.run(): This runs the GA for the specified number of generations, evolving the population toward better solutions by selecting the best features based on the fitness function. **Extracting the Best Solution**: best\_solution, best\_solution\_fitness, \_ = ga\_instance.best\_solution(): This retrieves the best feature selection solution after all generations have run. best\_solution contains the best feature set (as a binary vector where 1 means the feature is selected and 0 means it is excluded), and best\_solution\_fitness gives the corresponding fitness score.

A screenshot of a computer

Description automatically generated

* **Applying the Best Solution**: selected\_features = [idx for idx, value in enumerate(best\_solution) if value == 1]: This extracts the indices of the features selected by the genetic algorithm.
* If any error occurs during the GA process, it is caught by the except block, and an error message is printed.

**Step 5: Model selection**

A screenshot of a computer

Description automatically generated

* Here we define five machine learning models (KNN, Decision Tree, MLP, Random Forest, Gradient Boosting) and sets up hyperparameter tuning using Grid Search for each model with predefined parameter grids.

**Step 6: Hyper-parameter tuning:**

A screenshot of a computer

Description automatically generated

* **Model Hyperparameter Tuning using Grid search:** This is a method for hyperparameter tuning, where different combinations of parameters (from param\_grid) are tested. The model is evaluated using cross-validation (cv=3), and the best hyperparameters are selected based on accuracy.

**Step 7: Model evaluation**

A screenshot of a computer

Description automatically generated

* The code trains multiple models (KNN, Decision Tree, MLP, Random Forest, Gradient Boosting), makes predictions on the test set, and evaluates their performance using accuracy, confusion matrix, and classification report. This helps assess and compare the models' effectiveness.